第 40 卷 第 1 期 计 算 机 工 程 2014 年 1 月
Vol.40 No.1 Computer Engineering January 2014

·专栏· 文章编号：1000—3428(2014)01—0001—05 文献标识码：A 中图分类号：TP391

一种基于请求大小的固态盘 I/O 调度算法

吴素贞[a]，陈晓兰[a]，毛 波[b]

(厦门大学 a. 信息科学与技术学院计算机科学系；b. 软件工程系，福建 厦门 361005)

摘 要：对于同类型的 I/O 请求，基于闪存固态盘的请求响应时间与请求大小基本呈线性比例关系，并且固态盘的读写性能具有非对称性。针对该特性，提出一种基于请求大小的固态盘 I/O 调度(SIOS)算法，从 I/O 请求平均响应时间的角度提高固态盘设备的 I/O 性能。根据读写性能的非对称性，对读写请求进行分组并且优先处理读请求。在此基础上首先处理等待队列中的小请求，从而减少队列中请求的平均等待时间。采用 SLC 和 MLC 2 种类型的固态盘进行实验，在 5 种测试负载的驱动下与 Linux 系统中的 3 种调度算法进行比较，对于 SLC 固态盘，SIOS 平均响应时间分别减少 18.4%、25.8%、14.9%、14.5%和 13.1%，而对于 MLC 固态盘，平均响应时间分别减少 16.9%、24.4%、13.1%、13.0%和 13.7%，结果表明，SIOS 能有效减少 I/O 请求的平均响应时间，提高固态盘存储系统的 I/O 性能。

关键词：固态盘；请求调度；非对称读/写；请求大小；响应时间

An I/O Scheduling Algorithm for Solid State Disk Based on Request Size

WU Su-zhen[a], CHEN Xiao-lan[a], MAO Bo[b]

(a. Department of Computer Science, School of Information Science and Technology; b. Department of Software Engineering, Xiamen University, Xiamen 361005, China)

【Abstract】The response times are linear with the request sizes for flash-based Solid State Disk(SSD) with the same request type. Moreover, the read performance and write performance of flash-based SSD are asymmetric. Based on these characteristics, this paper proposes a Size-based I/O Scheduler(SIOS) for flash-based SSD to improve the I/O performance of SSD-based storage systems from the viewpoint of average response time. SIOS utilizes the asymmetric read and write performance characteristics of flash-based SSD and gives higher priority to the read requests. Moreover, by first processing the small requests in the I/O waiting queue, the average waiting times of the requests are reduced significantly. It implements SIOS in the Linux kernel and evaluates it with two kinds of SSD devices(SLC and MLC)driven by the five traces. Compared with the existing Linux disk I/O schedulers, evaluation results show that SIOS reduces average response times by 18.4%, 25.8%, 14.9%, 14.5% and 13.1% for SLC-based flash SSD, and reduces average response times by 16.9%, 24.4%, 13.1%, 13.0% and 13.7% for MLC-based flash SSD. Results show that compared with the state-of-the-arts, SIOS reduces the average response times significantly. Consequently, the I/O performance of the SSD-based storage systems is improved.

【Key words】Solid State Disk(SSD); request scheduling; asymmetric read/write; request size; response time

DOI: 10.3969/j.issn.1000-3428.2014.01.001

1 概述

基于闪存固态盘(Solid State Disk, SSD)的出现给存储系统带来了巨大的变化，由于其不同于磁盘的工作机理，固态盘存储系统的 I/O 性能相比磁盘存储系统有了极大的提高。由于固态盘由半导体器件组成，存储系统的可靠性和能效也得到了大幅度的提升[1]。近几年来，固态盘在各个领域都有广泛的应用，许多笔记本厂商已开始大量使用固态盘，而且如 Google 和百度等数据中心也已部署了基于固态盘的存储系统。然而现有的存储系统软件，如 I/O 调度算法等大多是基于磁盘设备设计实现的，将固态盘直接替换磁盘设备不能充分发挥固态盘存储设备的性能优势[2]。

I/O 调度算法是影响存储设备性能的一个重要因素。多个请求按照什么样的次序服务，直接影响存储设备的工作效率，进而影响整个存储系统的性能。传统的磁盘 I/O 调度算法都是基于请求的访问地址排序，尽可能让磁盘的磁头顺序地服务所有的请求。由于磁盘设备是一种机械式装置，通过磁头移动访问数据，因此请求的响应时间与请求的访

基金项目：国家自然科学基金青年科学基金资助项目“重复数据删除存储系统的数据重构性能和能效研究”(61100033)

作者简介：吴素贞(1982－)，女，助理教授、博士，主研方向：磁盘阵列存储系统，固态盘技术，重复数据删除技术；陈晓兰，硕士研究生；毛 波(通讯作者)，助理教授、博士

收稿日期：2013-07-12 **修回日期：**2013-09-02 **E-mail：**maobo@xmu.edu.cn

问地址有着密切的关系[3]。顺序访问可有效地减少磁头的寻道移动开销，提高磁盘的服务效率。如磁盘电梯调度算法优先考虑在磁道前进方向上最短时间，避免磁头在盘面上反复移动带来不必要的时间开销。但固态盘没有类似于磁盘的磁头机械装置，在读写操作中完全通过电路传输信号，不会存在类似磁盘的移动磁头、旋转盘片等机械动作。因此，传统针对磁盘的 I/O 调度算法已经不能适用于固态盘。

随着固态盘的广泛应用，传统的请求调度算法面临着挑战，人们根据固态盘的设备特性提出了新的 I/O 调度算法。现有的固态盘 I/O 调度算法主要分为 2 类：第 1 类算法主要为了权衡固态盘的 I/O 效率与服务的公平性，比如 Park S 等人提出的 FIOS[4]和 FlashFQ[5]调度算法；第 2 类算法主要是挖掘固态盘内部的并行性[6-8]，如 ParDispatcher 调度算法[9]。这些固态盘 I/O 调度算法都是从固态盘的某个特定方面对固态盘的 I/O 调度算法进行研究，但是都没有考虑固态盘的响应时间与请求大小之间的关系对 I/O 调度算法的影响，这正是固态盘 I/O 调度算法区别于磁盘 I/O 调度算法的一个非常重要的因素。

固态盘的性能特性对固态盘的 I/O 调度算法设计有着直接的影响。对于固态盘来说，请求的响应时间与请求大小基本上呈线性增长的关系。此外，由于闪存介质的特性，基于闪存的固态盘读写性能也不一样，读性能明显高于写性能[10]。针对固态盘这些性能特性，本文设计并实现一种基于请求大小的固态盘 I/O 调度(Size-based I/O Scheduler, SIOS)算法。该算法根据读写性能的非对称性，将读写请求分组后优先处理读请求，并在此基础上优先处理等待队列里的小请求，从而减少队列中请求的平均等待时间。

2 背景介绍

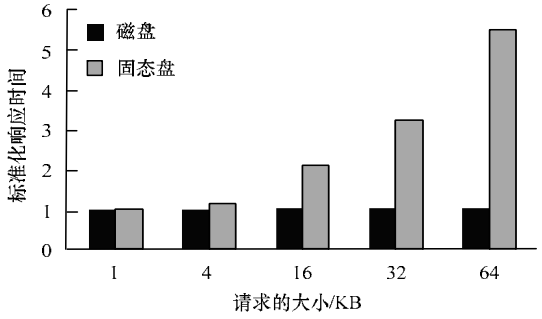
2.1 固态盘介绍

固态盘是一种新型的存储设备，与磁盘相比，具有读写速度更快、无噪声、能耗低、体积小等优势。基于闪存的固态盘采用闪存芯片作为存储介质，闪存中包含多个块，每个块中又包含多个页，每个页中一部分空间用于存储数据，还有一小部分空间用于存储元数据。由于闪存的介质特点，在对现有数据进行更新时必须先进行擦除操作，擦除操作以块为单位，而读写的基本单位则是页[11-12]。由于擦除的粒度比写入的粒度大，如果想保留块中的有效数据，需要在擦除前先读出该块中的有效数据，并在修改后写回，因此存在较差的随机写性能和介质损耗问题[13-14]。

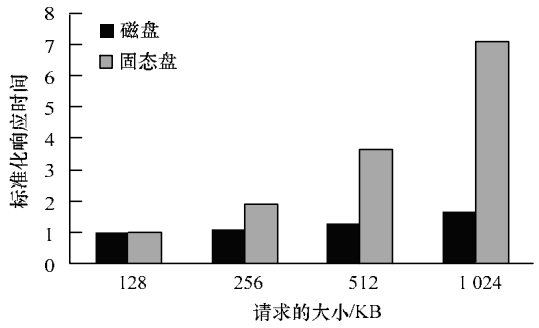
目前，固态盘已经成为存储系统中极其重要的一个组成部分，虽然磁盘是当前存储系统中的主要存储介质，但是磁盘存在速度慢、能耗大等性能瓶颈。由于磁盘固有的机械特性不能从根本上解决这些瓶颈问题，因此有必要用其他存储设备，如固态盘来替代或者辅助磁盘存储设备。

2.2 响应时间与请求大小的关系

为了验证磁盘和固态盘下请求的响应时间与请求大小之间的关系，本文采用 IOmeter 测试工具测试不同请求大小下磁盘(WDC WD1600AAJS)和固态盘(Intel X25-E 64 GB)的平均响应时间，图 1 为标准化响应时间的比较。实验结果表明，对于磁盘而言，请求的响应时间与请求大小关系不大。如图 1(a)所示，对于请求大小为 1 KB 到 64 KB 的请求，磁盘的响应时间几乎没有变化。这是因为磁盘通过寻道和旋转盘片进行定位和存取数据，访问磁盘的时间由三部分构成，分别是寻道时间、旋转等待延迟和数据传输时间。对于小的随机请求而言，寻道时间和旋转延迟占据了请求响应时间的主要因素。但当请求变大后，数据传输时间逐渐成为占据响应时间的重要组成部分，因此，平均响应时间就随着请求大小的增大而变长，如图 1(b)所示。其中，磁盘和固态盘的响应时间分别对应于各自的基准响应时间，其中，磁盘在 1 KB 和 128 KB 的响应时间分别是 13.3 ms 和 14.7 ms，固态盘在 1 KB 和 128 KB 的响应时间分别是 0.05 ms 和 0.54 ms。



(a)1 KB~32 KB 请求的性能比较



(b)64 KB~1 MB 请求的性能比较

图 1 不同请求大小随机访问下的标准化响应时间比较

此外，从图 1 可以看到，固态盘的请求平均响应时间与请求大小基本呈线性增长关系。这是因为固态盘没有类似于磁盘的机械装置，在读写操作中完全通过电路来传输信号，不会存在类似磁盘的移动磁头、旋转盘片等动作，所以数据传输时间是请求响应时间的主要部分。而数据传输时间与请求大小直接相关，因此，随着请求块大小的增大，固态盘请求的响应时间也随之线性增大。

在实验中，注意到无论对于多大的块，读请求的响应时间明显小于写请求的响应时间。这是因为固态盘多是采用基于闪存的存储介质，写操作往往需要首先完成以块为单位的擦除操作，所以固态盘的读写速率差异比较大。正

是基于固态盘的这种读写性能非对称性和请求大小对固态盘响应时间的影响，本文根据请求类型和请求大小来设计基于固态盘的 I/O 请求调度算法。

3　SIOS 调度算法

I/O 调度模块工作在块设备层和设备驱动层之间，按照一定的策略决定请求被服务的顺序。如图 2 所示为 SIOS 调度算法在整个 I/O 子系统中的位置及其系统结构。对于上层来说，请求的数据进入 I/O 调度模块，相当于一个入队的操作，之后根据一定的调度策略决定请求如何排队。对于下层来说，相当于一个出队的操作，调度策略选择下一个要服务的请求。SIOS 调度算法一方面要提高系统的吞吐率，另一方面需要降低系统的平均响应时间。

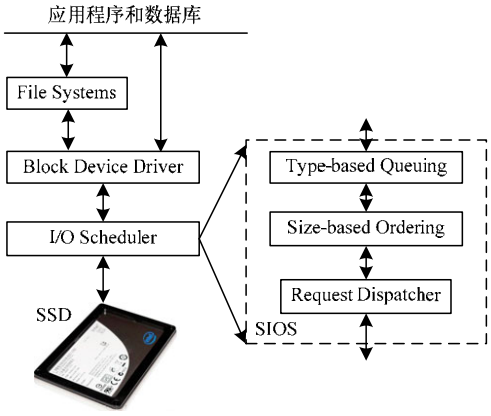


图 2　SIOS 调度算法的系统结构

SIOS 调度算法的主要目标是使得整个固态盘存储系统的平均响应时间最短，主要包含 3 部分：基于请求类型的排队(Type-based Queuing)，基于请求大小的排序(Size-based Ordering)和请求分发模块(Request Dispatcher)。基于请求类型的排队模块主要是执行读优先策略将读写请求分开入队列，而基于请求大小的排序主要是执行小请求优先策略，在读写队列中对请求进行排序。此外为防止请求饥饿现象的发生，SIOS 调度算法设置超时时间戳和写被阻塞的阈值。

3.1　基于请求类型的排队

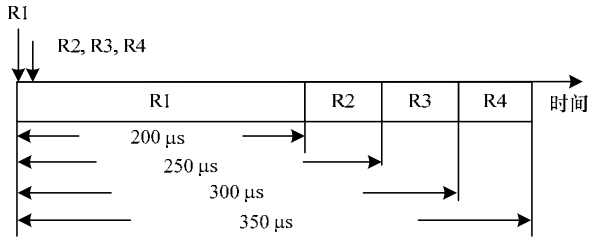
固态盘的读写性能是非对称的，读性能要比写性能好很多。此外，在上层应用中读操作是同步的，上层应用要等到读请求的响应数据返回后才能继续处理；而写操作是异步的，不会阻塞应用。如果读请求被写请求阻塞的话，将会影响整个应用程序。SIOS 算法采用读请求优先于写请求的方法，避免了读请求被写请求长时间阻塞，减少读请求的等待时间。基于请求类型的排队模块根据进入队列中请求的类型，将请求插入到对应类型的请求队列。

在读请求优先于写请求的情况下，当请求队列中有读请求就优先处理读请求，只有当队列中没有读请求时才去处理写请求。这种处理方法可以保证读请求不会被写请求阻塞，但是有可能造成写请求一直被读请求阻塞而长期得不到处理。为了防止这种写饥饿现象的发生，基于请求类型的排队模块在将写请求插入到写请求队列中时设置每个写请求被阻塞的次数，当阻塞次数达到预设的阈值时，说明该写请求不能再继续被阻塞，SIOS 调度算法将立即处理阻塞次数已达到阈值的写请求。这样就保证了在读优先的情况下，不会造成写饥饿的现象发生。

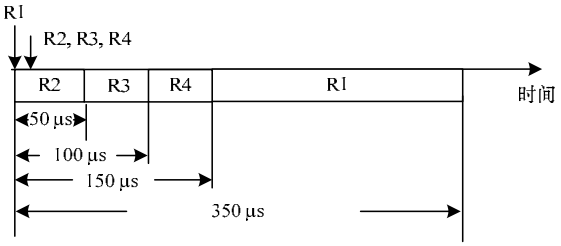
3.2　基于请求大小的排序

固态盘的请求响应时间与请求大小基本呈线性增长关系，越小请求的响应时间越短。优先处理小请求，然后再处理大请求，可以减少小请求在队列中的等待时间。因为如果小请求被大请求阻塞，小请求等待大请求完成的时间可能比其本身处理所需的时间长。如果优先处理小请求，由于小请求处理时间很短，对大请求响应时间的影响不大。

基于请求大小的排序模块在读写队列内部根据请求大小对请求进行排序，请求分发模块根据排序后的顺序将请求从对应队列中取出并发向固态盘。基于请求大小排序的固态盘调度算法可有效地减少请求在队列中的等待时间，从而减少固态盘的平均响应时间。如图 3 所示，有 1 个 16 KB 请求 R1(完成每个请求所需时间为 200 μs)和 3 个 4 Byte 请求 R2、R3 和 R4(完成每个请求所需时间为 50 μs)几乎同时到达(R1 请求先到达)，按照先来先服务的策略优先服务大请求，则处理 4 个请求的时间为 1 100 μs，存储系统的平均响应时间为 275 μs，如图 3(a)所示。如优先处理小请求，则所需时间为 750 μs，平均响应时间为 150 μs，相比之下存储系统平均响应时间减少 45.5%，如图 3(b)所示。



(a)Noop 调度算法服务请求的示意图



(b)SIOS调度算法服务请求的示意图

图 3　基于请求调度算法与传统调度算法的比较

优先处理小请求减少整个存储系统的平均响应时间，但也有可能会造成大请求一直等待小请求而得不到处理，即发生大请求饿死的现象。为避免这种情况的发生，SIOS 调度算法对读写请求都分别设置不同的请求超时时间戳，当请求在队列中等待的时间达到预设时间戳时，SIOS 调度算法立即处理该请求而不再等待。在选择请求类型中，同样遵循读优先的策略，所以优先服务读请求队列中的请求。

3.3 算法流程

SIOS 固态盘请求调度算法的流程如图 4 所示。读写请求分别在不同的队列中，根据队列中的请求情况，首先确定下一个请求的类型(读或写)，然后根据读写队列是否为空以及是否有写饥饿请求等待来判断。之后根据请求的类型判断在相应的请求队列中是否有超时的请求，如果有则立即处理该超时请求，如果没有则获取请求队列中最小的请求执行 I/O 操作。当队列中没有读写请求时，I/O 调度程序将一直等待直到有请求到达，否则就循环执行直到队列中所有的请求，直至被执行完。

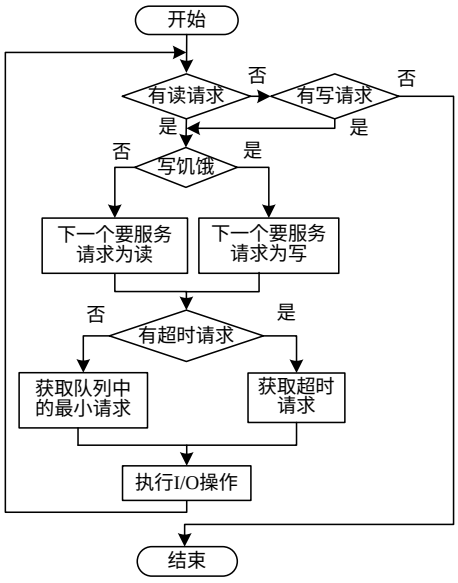


图 4 SIOS 调度算法的流程

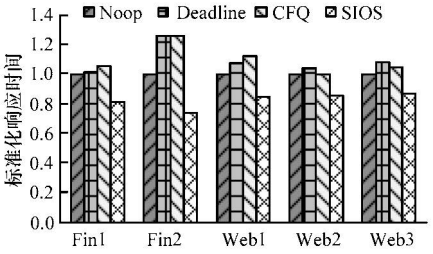
4 实验结果与分析

本文在 Linux 2.6.38 内核上进行测试，采用 Intel Core 3.00 GHz 的处理器和 4 GB 内存，固态盘采用 Intel® X25-E Extreme SATA Solid-State Drive 64 GB(以下称为 Intel X25-E 固态盘)和 Intel® Solid-State Drive 320 Series 300 GB(以下称为 Intel 320 固态盘)。实验中采用的负载为 2 个在线事务处理负载(Fin1 和 Fin2)和 3 个来自搜索引擎的负载(Web1、Web2 和 Web3)，负载的属性如表 1 所示。为了更好地模拟固态盘的应用负载，所有负载的每秒输入输出操作(IOPS)都被增强到 3 000 左右。实验中将 SIOS 调度算法与 Linux 系统中的 Noop、Deadline 和 CFQ 调度算法进行比较，比较的指标是请求的平均响应时间。

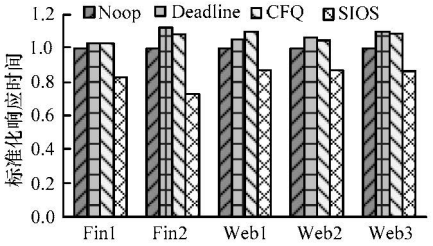
表 1 负载属性

负载	请求大小/Byte	读比例/(%)	IOPS
Fin1	512~17 116 160	21.6	111
Fin2	512~262 656	82.4	108
Web1	512~1 137 664	99.9	335
Web2	8 192~32 768	99.9	331
Web3	512~23 674 880	99.9	218

在上述 5 种负载驱动下，图 5(a)为 Intel X25-E 固态盘的测试结果，平均响应时间分别减少了 18.4%、25.8%、14.9%、14.5%和 13.1%；图 5(b)为 Intel 320 固态盘的测试结果，平均响应时间分别减少了 16.9%、24.4%、13.1%、13.0%和 13.7%。在不同负载中，读写请求所占的比例不同，这 5 种负载中 Fin1 和 Fin2 是读写混合型的应用，而由于写请求要比读请求慢很多，因此在负载 Fin1 和 Fin2 中读请求会花比较长的时间去等待写请求的完成。在这种情况下，设置读优先策略减少了请求的等待时间，而对于写请求来说，由于处理读请求比处理写请求快许多，因此对于写请求不会有很大的影响。此外，通过基于请求大小的排序，在读写请求队列内部优先处理小请求减少了小请求在队列中的等待时间，从而减少了整个存储系统的请求响应时间。



(a)Intel X25-E 固态盘的测试结果



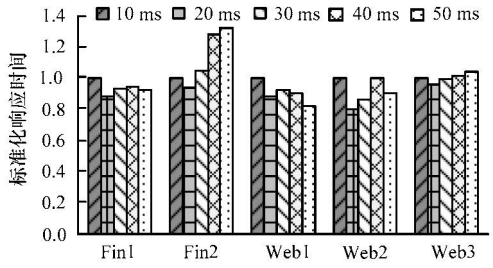
(b)Intel 320 固态盘的测试结果

图 5 各种调度算法性能测试结果比较

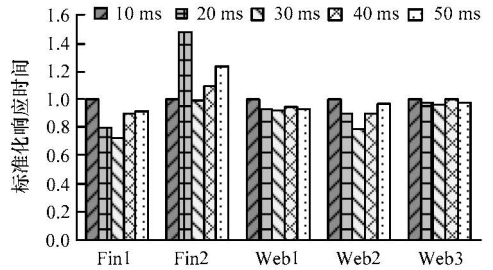
从图 5 还可以发现，负载 Web1、Web2 和 Web3 为读请求密集型负载，基本上不会发生读被写阻塞的情况，同时这些请求大小基本上集中在 8 KB 到 32 KB 之间并且往往相邻的几个请求大小是相同的，通过基于请求大小的排序，平均响应时间减少了 13%~15%。通过图 5(a)和 5(b)的对比，可以看到 SIOS 调度算法对不同的固态盘都是有效的，因此具有较好的可适用性。

为了确定不同的时间戳对测试结果的影响，本文针对不同的超时时间戳进行了测试。图 6 给出了不同的读请求超时时间戳下测试 SIOS 算法的平均响应时间结果，图 6(a)为 Intel X25-E 固态盘的测试结果，图 6(b)为 Intel 320 固态盘的测试结果。对于每个测试文件，分别设置了 5 种不同的时间戳(10 ms~50 ms)，测试不同的时间戳对平均响应时间的影响。从图 6 中可以看到，无论对于哪个测试文件，不同的超时时间戳之间的差别并不大。但是对于 Intel X25-E 固态盘，在设置超时时间戳为 20 ms 时系统的响应时间是最小的，如图 6(a)所示。而对于 Intel 320 固态盘而言，设

置超时时间戳为 30 ms 时系统的响应时间是最小的，如图 6(b)所示。测试结果说明对于性能较好的固态盘应该设置比较小的超时时间戳。



(a)Intel X25-E 固态盘的测试结果



(b)Intel 320 固态盘的测试结果

图 6 不同时间戳下 SIOS 调度算法的平均响应时间比较

在测试中还可以发现，设置不同的超时时间戳时发生超时请求的数量是不同的。当超时时间戳在 10 ms 以下时，超时请求的数量大概占到总请求数量的千分之一左右；当超时时间戳为 20 ms 时只有很少数的请求发生超时；超时时间戳为 30 ms 以上就基本上没有请求发送超时了。虽然当超时时间戳比较小时发生超时的请求比较多，但是由于请求所等待的时间比较短，因此整个存储系统的响应时间相差并不会太大。

5 结束语

磁盘的读写速度与 CPU 的处理速度相比，慢了多个数量级，I/O 性能已经成为数据密集型应用的性能瓶颈，而固态盘的出现在一定程度上缓解了这一问题。对于存储系统而言，I/O 调度策略对于各个读写请求担当了裁判的角色，以尽可能获取最优的 I/O 性能。一个合适的 I/O 调度算法能够提高整个存储系统的性能。本文分析了固态盘不同于磁盘的读写特性，在此基础上利用固态盘的响应时间与请求大小呈正比例的关系，提出一种基于请求大小的固态盘 I/O 调度(SIOS)算法，以减少固态盘的平均响应时间。SIOS 调度算法分离读写请求，从请求大小与响应时间的关系考虑优先执行小请求，从而有效地减少了请求在队列中的平均等待时间。SIOS 调度算法同时考虑大请求的饥饿问题，针对读写请求分别设置了超时时间戳。本文对 2 种不同固态盘设备的性能进行测试，结果表明，SIOS 调度算法有效地减少了请求的平均响应时间，提高固态盘存储系统的 I/O 性能。

参考文献

[1] Agrawal N, Prabhakaran V, Wobber T, et al. Design Tradeoffs for SSD Performance[C]//Proc. of 2008 USENIX Annual Technical Conference. Boston, USA: [s. n.], 2008.

[2] Dirik C, Jacob B. The Performance of PC Solid-state Disks as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization[C]//Proc. of the 36th International Symposium on Computer Architecture. Austin, USA: IEEE Press, 2009.

[3] Ruemmler C, Wilkes J. An Introduction to Disk Drive Modeling[J]. IEEE Computer, 1994, 27(3):17-29.

[4] Park S, Shen Kai. FIOS: A Fair, Efficient Flash I/O Scheduler[C]//Proc. of the 10th USENIX Conference on File and Storage. San Jose, USA: [s. n.], 2012.

[5] Shen Kai, Park S, FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs[C]//Proc. of USENIX Annual Technical Conference. San Jose, USA: [s. n.], 2013.

[6] Chen Feng, Lee R, Zhang Xiaodong. Essential Roles of Exploiting Internal Parallelism of Flash Memory Based Solid State Drives in High-speed Data Processing[C]//Proc. of the 17th International Symposium on High-performance Computer Architecture. San Francisco, USA: IEEE Press, 2011.

[7] Hu Yang, Jiang Hong, Feng Dan. Exploring and Exploiting the Multi-level Parallelism Inside SSD for Improved Performance and Endurance[J]. IEEE Transactions on Computers, 2013, 62(6): 1141-1155.

[8] 范玉雷, 赖文豫, 孟小峰. 基于固态硬盘内部并行的数据库表扫描与聚集[J]. 计算机学报, 2012, 35(11): 2327-2336.

[9] Wang Hua, Huang Ping, He Shuang, et al, A Novel I/O Scheduler for SSD with Improved Performance and Lifetime[C]//Proc. of the 29th IEEE Conference on Massive Data Storage. Long Beach, USA: IEEE Press, 2013.

[10] Chen Feng, Koufaty D A, Zhang Xiaodong. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives[C]//Proc. of 2009 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Seattle, USA: ACM Press, 2009.

[11] 郑文静, 李明强, 舒继武. Flash 存储技术[J]. 计算机研究与发展, 2010, 47(4): 716-726.

[12] Mao Bo, Jiang Hong, Wu Suzhen, et al, HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability[J]. ACM Transactions on Storage, 2012, 8(1): 1-22.

[13] Min C, Kim K, Cho H, et al. SFS: Random Write Considered Harmful in Solid State Drives[C]//Proc. of the 10th USENIX Conference on File and Storage Technologies. San Jose, USA: [s. n.], 2012.

[14] 吴素贞, 陈晓熹, 毛 波. GC-RAIS: 一种基于垃圾回收感知的固态盘阵列[J]. 计算机研究与发展, 2013, 50(1): 60-68.

编辑 索书志